CARMA: Collective Adaptive Resource-sharing Markovian Agents

CARMA: Collective Adaptive Resource-sharing Markovian Agents

Luca Bortolussi
Saarland University
University of Trieste
ISTI - CNR

Rocco De Nicola
IMT Lucca

Vashti Galpin
University of Edinburgh

Stephen Gilmore
University of Edinburgh

Jane Hillston
University of Edinburgh

Diego Latella
ISTI - CNR

Michele Loreti
Università di Firenze
IMT Lucca

Mieke Massink
ISTI - CNR

In this paper we present CARMA, a language recently defined to support specification and analysis of collective adaptive systems. CARMA is a stochastic process algebra equipped with linguistic constructs specifically developed for modelling and programming systems that can operate in open-ended and unpredictable environments. This class of systems is typically composed of a huge number of interacting agents that dynamically adjust and combine their behaviour to achieve specific goals. A CARMA model, termed a *collective*, consists of a set of components, each of which exhibits a set of attributes. To model dynamic aggregations, which are sometimes referred to as *ensembles*, CARMA provides communication primitives that are based on predicates over the exhibited attributes. These predicates are used to select the participants in a communication. Two communication mechanisms are provided in the CARMA language: multicast-based and unicast-based. In this paper, we first introduce the basic principles of CARMA and then we show how our language can be used to support specification with a simple but illustrative example of a socio-technical collective adaptive system.

1 Introduction

Collective adaptive systems (CAS) typically consist of very large numbers of components which exhibit autonomic behaviour depending on their properties, objectives and actions. Decision-making in such systems is complicated and interaction between their components may introduce new and sometimes unexpected behaviours. CAS are open, in the sense that components may enter or leave the collective at anytime. Components can be highly heterogeneous (machines, humans, networks, etc.) each operating at different temporal and spatial scales, and having different (potentially conflicting) objectives. We are still far from being able to design and engineer real collective adaptive systems, or even specify the principles by which they should operate.

CAS thus provide a significant research challenge in terms of both representation and reasoning about their behaviour. The pervasive yet transparent nature of the applications developed in this paradigm makes it of paramount importance that their behaviour can be thoroughly assessed during their design, prior to deployment, and throughout their lifetime. Indeed their adaptive nature makes modelling essential and models play a central role in driving their adaptation. Moreover, the analysis should encompass both functional and non-functional aspects of behaviour. Thus it is vital that we have available robust modelling techniques which are able to describe such systems and to reason about their behaviour in both qualitative and quantitative terms. To move towards this goal, we consider it important to develop a theoretical foundation for collective adaptive systems that would help in understanding their distinc-

Submitted to:
QAPL 2015

tive features. In this paper we present CARMA, a language designed within the QUANTICOL project[1] specifically for the specification and analysis of CAS, with the particular objective of supporting quantitative evaluation and verification.

CARMA builds on a long tradition of stochastic process algebras such as PEPA [13], MTIPP [12], EMPA [2], Stochastic π-Calculus [15], Bio-PEPA [5], MODEST [3] and others [11, 4]. It combines the lessons which have been learned from these languages with those learned from developing languages to model CAS, such as SCEL [8] and PALOMA [9], which feature attribute-based communication and explicit representation of locations.

SCEL [8] (Software Component Ensemble Language), is a kernel language that has been designed to support the programming of autonomic computing systems. This language relies on the notions of *autonomic components* representing the collective members, and *autonomic-component ensembles* representing collectives. Each component is equipped with an interface, consisting of a collection of attributes, describing different features of components. Attributes are used by components to dynamically organise themselves into ensembles and as a means to select partners for interaction. The stochastic variant of SCEL, called StocS [14], was a first step towards the investigation of the impact of different stochastic semantics for autonomic processes, that relies on stochastic output semantics, probabilistic input semantics and on a probabilistic notion of knowledge. Moreover, SCEL has inspired the development of the core calculus AbC [1] that focuses on a minimal set of primitives that defines attribute-based communication, and investigates their impact. Communication among components takes place in a broadcast fashion, with the characteristic that only components satisfying predicates over specific attributes receive the sent messages, provided that they are willing to do so.

PALOMA [9] is a process algebra that takes as starting point a model based on located Markovian agents each of which is parameterised by a location, which can be regarded as an attribute of the agent. The ability of agents to communicate depends on their location, through a perception function. This can be regarded as an example of a more general class of attribute-based communication mechanisms. The communication is based on a multicast, as only agents who enable the appropriate reception action have the ability to receive the message. The scope of communication is thus adjusted according to the perception function.

A distinctive contribution of the new language is the rich set of communication primitives that are offered. CARMA supports both unicast and broadcast communication, and locally synchronous, but globally asynchronous communication. This richness is important to enable the spatially distributed nature of CAS, where agents may have only local awareness of the system, yet the design objectives and adaptation goals are often expressed in terms of global behaviour. Representing these rich patterns of communication in classical process algebras or traditional stochastic process algebras would be difficult, and would require the introduction of additional model components to represent buffers, queues and other communication structures. Another feature of CARMA is the explicit representation of the environment in which processes interact, allowing rapid testing of a system under different open world scenarios. The environment in CARMA models can evolve at runtime, due to the feedback from the system, and it further modulates the interaction between components, by shaping rates and interaction probabilities. Furthermore the large scale nature of CAS systems makes it essential to support scalable analysis techniques, thus CARMA has been designed anticipating both a discrete and a continuous semantics in the style of [16].

The focus of this paper is the presentation of the language and its discrete semantics, which are presented in the FUTS style [7]. The structure of the paper is as follows. Section 2 presents the syntax

[1] `http://www.quanticol.eu`

of the language and explains the organisation of a model in terms of a collective of agents that are considered in the context of an environment. In Section 3 we give a detailed account of the semantics, particularly explaining the role of the environment. The use of CARMA is illustrated in Section 4 where we describe a model of a simple bike sharing system. Some conclusions are drawn in Section 5.

2 CARMA syntax

A CARMA system consists of a *collective* (N) operating in an *environment* ($\mathscr{E}$). The collective consists of a set of components. It models the behavioural part of a system and is used to describe a set of interacting *agents* that cooperate to achieve a set of given tasks. The environment models all those aspects which are intrinsic to the context where the agents under consideration are operating. The environment also mediates agent interactions.

We let SYS be the set of CARMA *systems* S defined by the following syntax:

$$S ::= N \textbf{ in } \mathscr{E}$$

where N is a collective and $\mathscr{E}$ is an environment. The latter provides the global state of the system and governs the interactions in the collective.

We let COL be the set of collectives N which are generated by the following grammar:

$$N ::= C \mid N \parallel N$$

A collective N is either a *component* C or the parallel composition of two collectives ($N \parallel N$).

A component C can be either the *inactive component*, which is denoted by $\mathbf{0}$, or a term of the form (P,γ), where P is a *process* and γ is a *store*. A term (P,γ) models an *agent* operating in the system under consideration: the process P represents the agent's behaviour whereas the store γ models its *knowledge*. A store is a function which maps *attribute names* to *basic values*. We let:

- ATTR be the set of *attribute names* a, a', a_1,..., b, b', b_1,...;
- VAL be the set of *basic values* v, v', v_1,...;
- Γ be the set of *stores* $\gamma, \gamma_1, \gamma', \ldots$ i.e. functions from ATTR to VAL.

We let COMP be the set of components C generated by the following grammar:

$$C ::= \mathbf{0} \mid (P,\gamma)$$

We let PROC be the set of processes P, Q,... defined by the following grammar:

$$\begin{array}{rcll} P,Q & ::= & \textbf{nil} & \\ & \mid & \textbf{kill} & \\ & \mid & act.P & \\ & \mid & P+Q & \\ & \mid & P \mid Q & \\ & \mid & [\pi]P & \\ & \mid & A & (A \stackrel{\triangle}{=} P) \end{array} \qquad \begin{array}{rcl} act & ::= & \alpha^\star[\pi]\langle\overrightarrow{e}\rangle\sigma \\ & \mid & \alpha[\pi]\langle\overrightarrow{e}\rangle\sigma \\ & \mid & \alpha^\star[\pi](\overrightarrow{x})\sigma \\ & \mid & \alpha[\pi](\overrightarrow{x})\sigma \\ e & ::= & a \mid \mathsf{this}.a \mid x \mid v \mid \cdots \\ \pi & ::= & \top \mid \bot \mid e_1 \bowtie e_2 \mid \neg\pi \mid \pi \wedge \pi \mid \cdots \end{array}$$

In CARMA processes can perform four types of actions: *broadcast output* ($\alpha^\star[\pi]\langle\overrightarrow{e}\rangle\sigma$), *broadcast input* ($\alpha^\star[\pi](\overrightarrow{x})\sigma$), *output* ($\alpha[\pi]\langle\overrightarrow{e}\rangle\sigma$), and *input* ($\alpha[\pi](\overrightarrow{x})\sigma$). Where:

- α is an *action type* in the set of action type ACTTYPE;
- π is an *predicate*;
- x is a *variable* in the set of variables VAR;
- $\overrightarrow{\cdot}$ indicates a sequence of elements;
- σ is an *update*, i.e. a function from Γ to $Dist(\Gamma)$ in the set of *updates* Σ; where $Dist(\Gamma)$ is the set of probability distributions over Γ.

The admissible communication partners of each of these actions are identified by the predicate π. This is a predicate on *attribute names*. Note that, in a component (P,γ) the store γ regulates the behaviour of P. Primarily, γ is used to evaluate the predicate associated with an action in order to filter the possible synchronisations involving process P. In addition, γ is also used as one of the parameters for computing the actual rate of actions performed by P. The process P can change γ immediately after the execution of an action. This change is brought about by the *update* σ. The update is a function that when given a store γ returns a probability distribution over Γ which expresses the possible evolutions of the store after the action execution.

The *broadcast output* $\alpha^\star[\pi]\langle\overrightarrow{e}\rangle\sigma$ models the execution of an action α that spreads the values resulting from the evaluation of expressions $\overrightarrow{e}$ in the local store γ. This message can be potentially received by any process located at components whose store satisfies predicate π. This predicate may contain references to attribute names that have to be evaluated under the local store. These references are prefixed by the special name this. For instance, if loc is the attribute used to store the position of a component, action

$$\alpha^\star[\mathsf{distance}(\mathsf{this.loc},\mathsf{loc}) \leq L]\langle\overrightarrow{v}\rangle\sigma$$

potentially involves all the components located at a distance that is less than or equal to a given threshold L. The *broadcast output* is non-blocking. The action is executed even if no process is able to receive the values which are sent. Immediately after the execution of an action, the update σ is used to compute the (possible) *effects* of the performed action on the store of the hosting component where the output is performed.

To receive a broadcast message, a process executes a *broadcast input* of the form $\alpha^\star[\pi](\overrightarrow{x})\sigma$. This action is used to receive a tuple of values $\overrightarrow{v}$ sent with an action α from a component whose store satisfies the predicate $\pi[\overrightarrow{v}/\overrightarrow{x}]$. The transmitted values can be part of the predicate π. For instance, $\alpha^\star[x>5](x)\sigma$ can be used to receive a value that is greater than 5.

The other two kinds of action, namely *output* and *input*, are similar. However, differently from broadcasts described above, these actions realise a *point-to-point* interaction. The *output* operation is blocking, in contrast with the non-blocking broadcast output.

Choice and parallel composition are the usual definitions for process algebras. Processes can be guarded so that $[\pi]P$ behaves as the process P if the predicate π is satisfied. Finally, process **kill** is used to *destroy* a component. We assume that this term always occurs under the scope of an action prefix.

CARMA collectives operate in an environment $\mathscr{E}$. This environment is used to model the intrinsic rules that govern, for instance, the physical context where our system is situated.

An environment consists of two elements: a *global store* γ_g, that models the overall state of the system, and an *evolution rule* ρ. The latter is a function which, depending on the global store and the current state of the collective, i.e. the configurations of each component in the collective, returns a tuple of functions $\varepsilon = \langle\mu_p,\mu_r,\mu_u\rangle$ known as the *evaluation context* where $\text{ACT} = \text{ACTTYPE} \cup \{\alpha^\star | \alpha \in \text{ACTTYPE}\}$ and:

- $\mu_p : \Gamma \times \text{ACT} \to [0,1]$, expresses the probability to receive a message;
- $\mu_r : \Gamma \times \Gamma \times \text{ACT} \to \mathbb{R}_{\geq 0}$, computes the execution rate of an action;
- $\mu_u : \Gamma \times \text{ACT} \to \Sigma \times \text{COL}$, determines the updates on the environment (global store and collective) induced by the action execution.

These functions regulate system behaviour. Function μ_p, which takes as parameters the local stores of the two interacting components, i.e. the sender and the receiver, and the action used to interact, returns the probability to receive a message. Function μ_r computes the rate of an unicast/broadcast output. This function takes as parameter the local store of the component performing the action and the action on which interaction is based. Note that the environment can disable the execution of a given action. This happens when the function μ_r (resp. μ_p) returns the value 0. Finally, the function μ_u is used to update the global store and to install a new collective in the system. The function μ_u takes as parameters the store of the component performing the action together with the action type and returns a pair (σ, N). Within this pair, σ identifies the update on the global store whereas N is a new collective installed in the system. This function is particularly useful for modelling the arrival of new agents into a system. All of these functions are determined by an *evolution rule* ρ depending on the global store and the actual state of the components in the system. For instance, the probability to receive a given message may depend on the *concentration* of components in a given state. Similarly, the actual rate of an action may be a function of the number of components whose store satisfies a given property.

3 CASPA operational semantics

In this section we define the operational semantics of CARMA specifications. This operational semantics is defined in three stages. First, we introduce the transition relation $\stackrel{\cdot}{\longrightarrow}_{\cdot}$ that describes the behaviour of a single component. Second, this relation is used to define the transition relation $\stackrel{\cdot}{\longrightarrow}_{\cdot}$ which describes the behaviour of collectives. Finally, the transition relation $\stackrel{\cdot}{\longmapsto}$ will be defined to show how CARMA systems evolve.

All these transition relations are defined in the FUTS style [7]. Using this approach, a transition relation is described using a triple of the form $(N, \ell, \mathcal{N})$. The first element of this triple is either a component, or a collective, or a system. The second element is a transition label. The third element is a function associating each component, collective, or system with a non-negative number. A non-zero value represents the rate of the exponential distribution characterising the time needed for the execution of the action represented by ℓ. The zero value is associated with unreachable terms. We use the FUTS style semantics because it makes explicit an underlying Action Labelled Markov Chain, which can be simulated with standard algorithms [10] but is nevertheless more compact than Plotkin-style semantics, as the functional form allows different possible outcomes to be treated within a single rule. A complete description of FUTS and their use can be found in [7].

3.1 Operational semantics of components

We use the transition relation $\rightharpoonup_{\varepsilon} \subseteq \text{COMP} \times \text{LAB} \times [\text{COMP} \to \mathbb{R}_{\geq 0}]$ to define the behaviour of a single component. In this relation $[\text{COMP} \to \mathbb{R}_{\geq 0}]$ denotes the set of functions from COMP to $\mathbb{R}_{\geq 0}$ and LAB is

the set of transition labels ℓ which are generated by the following grammar:

$$\begin{array}{rcll} \ell & ::= & \alpha^\star[\pi]\langle\overrightarrow{v}\rangle,\gamma & \text{Broadcast output} \\ & | & \alpha^\star[\pi](\overrightarrow{v}),\gamma & \text{Broadcast input} \\ & | & \alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma & \text{Unicast Output} \\ & | & \alpha[\pi](\overrightarrow{v}),\gamma & \text{Unicast Input} \\ & | & \tau[\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma] & \text{Unicast Synchronization} \\ & | & \mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}),\gamma] & \text{Broadcast Input Refusal} \end{array}$$

The first four labels are associated with the four CARMA input-output actions and they contain a reference to the action which is performed (α or $\alpha^\star$), the store of the component where the action is executed (γ), and the value which is transmitted or received. The transition label $\tau[\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma]$ is the one which is associated with *unicast synchronisation*. The final label $\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}),\gamma]$ denotes the case where a component is not able to receive a broadcast output. This arises at the level of the single component either because the associated message has been lost, or because no process is willing to receive that message. We will observe later in this section that the use of $\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}),\gamma]$ labels are crucial to handle appropriately *dynamic process operators*, namely *choice* and *guard*.

The transition relation $\multimap_\varepsilon$, as formally defined in Table 1 and Table 2, is parametrised with respect to an *evaluation context* ε. This is used to compute the actual rate of process actions and to compute the probability to receive messages.

The process **nil** denotes the process that cannot perform any action. The transitions which are induced by this process at the level of components can be derived via rules **Nil** and **Nil-F1**. These rules respectively say that the inactive process cannot perform any action, and always refuses any broadcast input. Note that, the fact that a component $(\mathbf{nil},\gamma)$ does not perform any transition is derived from the fact that any label that is not a *broadcast input refusal* leads to function $\emptyset$ (rule **Nil**). Indeed, $\emptyset$ denotes the 0 constant function. Conversely, **Nil-F1** states that $(\mathbf{nil},\gamma)$ can always perform a transition labelled $\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}),\gamma]$ leading to $[(\mathbf{nil},\gamma)\mapsto 1]$, where $[C\mapsto v]$ denotes the function mapping the component C to $v\in\mathbb{R}_{\geq 0}$ and all the other components to 0.

The behaviour of a *broadcast output* $(\alpha^\star[\pi_1]\langle\overrightarrow{e}\rangle\sigma.P,\gamma)$ is described by rules **B-Out**, **B-Out-F1** and **B-Out-F2**. Rule **B-Out** states that a broadcast output $\alpha^\star[\pi]\langle\overrightarrow{e}\rangle\sigma$ can affect components that satisfy $\pi'=[\![\pi]\!]_\gamma$[2]. The action rate is determined by the evaluation context $\varepsilon=\langle\mu_p,\mu_r,\mu_u\rangle$ and, in particular, by the function μ_r. This function, given a store γ and the kind of action performed, in this case $\alpha^\star$, returns a value in $\mathbb{R}_{\geq 0}$. If this value is greater than 0, it denotes the execution rate of the action. However, the evaluation context can disable the execution of some actions. This happens when $\mu_r(\gamma,\alpha^\star)=0$. The possible next local stores after the execution of an action are determined by the update σ. This takes the store γ and yields a probability distribution $\mathbf{p}=\sigma(\gamma)\in \mathit{Dist}(\Gamma)$. In rule **B-Out**, and in the rest of the paper, the following notations are used:

- let $P\in$ PROC and $\mathbf{p}\in\mathit{Dist}(\Gamma)$, $(P,\mathbf{p})$ is a probability distribution in $\mathit{Dist}(\text{COMP})$ such that:

$$(P,\mathbf{p})(C)=\begin{cases} 1 & P\equiv Q|\mathbf{kill}\ \wedge\ C\equiv\mathbf{0} \\ \mathbf{p}(\gamma) & C\equiv(P,\gamma)\ \wedge\ P\not\equiv Q|\mathbf{kill} \\ 0 & \text{otherwise} \end{cases}$$

- let $\mathbf{c}\in\mathit{Dist}(\text{COMP})$ and $r\in\mathbb{R}_{\geq 0}$, $r\cdot\mathbf{c}$ denotes the function $\mathscr{C}:\text{COMP}\to\mathbb{R}_{\geq 0}$ such that: $\mathscr{C}(C)=r\cdot\mathbf{c}(C)$

[2] We let $[\![\cdot]\!]_\gamma$ denote the evaluation function of an expression/predicate with respect to the store γ.

$$\frac{\ell \neq \mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}),\gamma]}{(\mathbf{nil},\gamma) \xrightarrow{\ell}_\varepsilon \emptyset}\ \textbf{Nil} \qquad \frac{}{(\mathbf{nil},\gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}),\gamma]}_\varepsilon [(\mathbf{nil},\gamma) \mapsto 1]}\ \textbf{Nil-F1}$$

$$\frac{[\![\pi]\!]_\gamma = \pi' \quad [\![\overrightarrow{e}]\!]_\gamma = \overrightarrow{v} \quad \mathbf{p} = \sigma(\gamma) \quad \varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle}{(\alpha^\star[\pi]\langle \overrightarrow{e} \rangle \sigma.P,\gamma) \xrightarrow{\alpha^\star[\pi']\langle \overrightarrow{v} \rangle,\gamma}_\varepsilon \mu_r(\gamma,\alpha^\star)\cdot(P,\mathbf{p})}\ \textbf{B-Out}$$

$$\frac{}{(\alpha^\star[\pi_1]\langle \overrightarrow{e} \rangle \sigma.P,\gamma) \xrightarrow{\mathscr{R}[\beta^\star[\pi_2](\overrightarrow{v}),\gamma]}_\varepsilon [(\alpha^\star[\pi_1]\langle \overrightarrow{e} \rangle \sigma.P,\gamma) \mapsto 1]}\ \textbf{B-Out-F1}$$

$$\frac{[\![\pi]\!]_\gamma = \pi' \quad [\![\overrightarrow{e}]\!]_\gamma = \overrightarrow{v} \quad \ell \neq \alpha^\star[\pi']\langle \overrightarrow{v} \rangle,\gamma \quad \ell \neq \mathscr{R}[\beta^\star[\pi'](\overrightarrow{v}_1),\gamma]}{(\alpha^\star[\pi]\langle \overrightarrow{e} \rangle \sigma.P,\gamma) \xrightarrow{\ell}_\varepsilon \emptyset}\ \textbf{B-Out-F2}$$

$$\frac{[\![\pi_2[\overrightarrow{v}/\overrightarrow{x}]]\!]_{\gamma_2} = \pi_2' \quad \gamma_1 \models \pi_2' \quad \gamma_2 \models \pi_1 \quad \mathbf{p} = \sigma[\overrightarrow{v}/\overrightarrow{x}](\gamma_2) \quad \varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle}{(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \xrightarrow{\alpha^\star[\pi_1](\overrightarrow{v}),\gamma_1}_\varepsilon \mu_p(\gamma_1,\gamma_2,\alpha^\star)\cdot(P[\overrightarrow{v}/\overrightarrow{x}],\mathbf{p})}\ \textbf{B-In}$$

$$\frac{[\![\pi_2[\overrightarrow{v}/\overrightarrow{x}]]\!]_{\gamma_2} = \pi_2' \quad \gamma_1 \models \pi_2' \quad \gamma_2 \models \pi_1 \quad \varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle}{(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \xrightarrow{\mathscr{R}[\alpha^\star[\pi_1](\overrightarrow{v}),\gamma_1]}_\varepsilon [(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \mapsto 1-\mu_p(\gamma_1,\gamma_2,\alpha^\star)]}\ \textbf{B-In-F1}$$

$$\frac{[\![\pi_2[\overrightarrow{v}/\overrightarrow{x}]]\!]_{\gamma_2} = \pi_2' \quad (\gamma_1 \not\models \pi_2' \text{ or } \gamma_2 \not\models \pi_1)}{(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \xrightarrow{\alpha^\star[\pi_1](\overrightarrow{v}),\gamma_1}_\varepsilon \emptyset}\ \textbf{B-In-F2}$$

$$\frac{\ell \neq \alpha^\star[\pi_1](\overrightarrow{v}),\gamma_1 \quad \ell \neq \mathscr{R}[\alpha^\star[\pi_1](\overrightarrow{v}),\gamma_1]}{(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \xrightarrow{\ell}_\varepsilon \emptyset}\ \textbf{B-In-F3}$$

$$\frac{\alpha \neq \beta}{(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \xrightarrow{\mathscr{R}[\beta^\star[\pi_1](\overrightarrow{v}),\gamma_1]}_\varepsilon [(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2) \mapsto 1]}\ \textbf{B-In-F4}$$

Table 1: Operational semantics of components (Part 1)

Note that, after the execution of an action a component can be destroyed. This happens when the continuation process after the action prefixing contains the term **kill**. For instance, by applying rule **B-Out** we have that: $(\alpha^\star[\pi_1]\langle v\rangle\sigma.(\mathbf{kill}|Q),\gamma) \xrightarrow{\alpha^\star[\pi_1]\langle v\rangle,\gamma}_{\varepsilon} [\mathbf{0}\mapsto r]$.

Rule **B-Out-F1** states that a *broadcast output* always refuses any *broadcast input*, while **B-Out-F2** states that a *broadcast output* can be only involved in labels of the form $\alpha^\star[\pi]\langle\overrightarrow{v}\rangle,\gamma$ or $\mathscr{R}[\beta^\star[\pi_2](\overrightarrow{v}),\gamma]$.

Transitions related to a broadcast input are labelled with $\alpha^\star[\pi_1](\overrightarrow{v}),\gamma_1$. There, γ_1 is the store of the component executing the output, α is the action performed, π_1 is the predicate that identifies the target components, while $\overrightarrow{v}$ is the sequence of transmitted values. Rule **B-In** states that a component $(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2)$ can perform a transition with this label when its store γ_2 satisfies the target predicate, i.e. $\gamma_2\models\pi_1$, and the component executing the action satisfies the predicate $\pi_2[\overrightarrow{v}/\overrightarrow{x}]$. The evaluation context $\varepsilon=\langle\mu_p,\mu_r,\mu_u\rangle$ can influence the possibility to perform this action. This transition can be performed with probability $\mu_p(\gamma_1,\gamma_2,\alpha^\star)$.

Rule **B-In-F1** models the fact that even if a component can potentially receive a broadcast message, the message can get lost according to a given probability regulated by the evaluation context, namely $1-\mu_p(\gamma_1,\gamma_2,\alpha^\star)$. Rule **B-In-F2** models the fact that if a component is not in the set of possible receivers ($\gamma_2\not\models\pi_1$) or the sender does not satisfy the expected requirements ($\gamma_1\not\models\pi_2'$) then the component cannot receive a broadcast message. Finally, rules **B-In-F3** and **B-In-F4** model the fact that $(\alpha^\star[\pi_2](\overrightarrow{x})\sigma.P,\gamma_2)$ can only perform a broadcast input on action α and that it always refuses input on any other action type $\beta\neq\alpha$, respectively.

The behaviour of *unicast output* and *unicast input* is defined by the first six rules of Table 2. These rules are similar to the ones already presented for broadcast output and broadcast input. The only difference is that both unicast output (**Out-F1**) and unicast input (**In-F1**) always refuse any broadcast input with probability 1. The other rules of Table 2 describe the behaviour of other process operators, namely *choice* $P+Q$, *parallel composition* $P|Q$, *guard* and *recursion*.

The term $P+Q$ identifies a process that can behave either as P or as Q. The rule **Plus** states that the components that are reachable by $(P+Q,\gamma)$, via a transition that is not a *broadcast input refusal*, are the ones that can be reached either by (P,γ) or by (Q,γ). In this rule we use $\mathscr{C}_1\oplus\mathscr{C}_2$ to denote the function that maps each term C to $\mathscr{C}_1(C)+\mathscr{C}_2(C)$, for any $\mathscr{C}_1,\mathscr{C}_2\in[\textsc{Comp}\to\mathbb{R}_{\geq 0}]$. At the same time, process $P+Q$ *refuses* a broadcast input when both the process P and Q do that. This is modelled by **Plus-F1**, where, for each $\mathscr{C}_1:\textsc{Comp}\to\mathbb{R}_{\geq 0}$ and $\mathscr{C}_2:\textsc{Comp}\to\mathbb{R}_{\geq 0}$, $\mathscr{C}_1+\mathscr{C}_2$ denotes the function that maps each term of the form $(P+Q,\gamma)$ to $\mathscr{C}_1((P,\gamma))\cdot\mathscr{C}_2((Q,\gamma))$, while any other component is mapped to 0. Note that, differently from rule **Plus**, when rule **Plus-F1** is applied operator $+$ is not removed after the transition. This models the fact that when a broadcast message is refused the choice is not resolved.

In $P|Q$ the two composed processes interleave for all the transition labels except for broadcast input refusal (**Par**). For this label the two processes synchronise (**Par-F1**). This models the fact that a message is lost when both processes refuse to receive it. In the rules the following notations are used:

- for each component C and process Q we let:

$$C|Q=\begin{cases}\mathbf{0} & C\equiv\mathbf{0}\\ (P|Q,\gamma) & C\equiv(P,\gamma)\end{cases}$$

  $Q|C$ is symmetrically defined.
- for each $\mathscr{C}:\textsc{Comp}\to\mathbb{R}_{\geq 0}$ and process Q, $\mathscr{C}|Q$ (resp. $Q|\mathscr{C}$) denotes the function that maps each term of the form $C|Q$ (resp. $Q|C$) to $\mathscr{C}(C)$, while the others are mapped to 0;
- for each $\mathscr{C}_1:\textsc{Comp}\to\mathbb{R}_{\geq 0}$ and $\mathscr{C}_2:\textsc{Comp}\to\mathbb{R}_{\geq 0}$, $\mathscr{C}_1|\mathscr{C}_2$ denotes the function that maps each term of the form $(P|Q,\gamma)$ to $\mathscr{C}_1((P,\gamma))\cdot\mathscr{C}_2((Q,\gamma))$, while the others are mapped to 0.

$$\frac{[\![\pi]\!]_\gamma = \pi' \quad [\![\overrightarrow{e}]\!]_\gamma = \overrightarrow{v} \quad \mathbf{p} = \sigma(\gamma) \quad \varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle}{(\alpha[\pi]\langle \overrightarrow{e} \rangle \sigma.P, \gamma) \xrightarrow{\alpha[\pi']\langle \overrightarrow{v} \rangle, \gamma}_\varepsilon \mu_r(\gamma, \alpha) \cdot (P, \mathbf{p})} \textbf{ Out}$$

$$\frac{}{(\alpha[\pi_1]\langle \overrightarrow{e} \rangle \sigma.P, \gamma_1) \xrightarrow{\mathscr{R}[\beta^\star[\pi_2](\overrightarrow{v}), \gamma_2]}_\varepsilon [(\alpha[\pi_1]\langle \overrightarrow{e} \rangle \sigma.P, \gamma_1) \mapsto 1]} \textbf{ Out-F1}$$

$$\frac{[\![\pi]\!]_\gamma = \pi' \quad [\![\overrightarrow{e}]\!]_\gamma = \overrightarrow{v} \quad \ell \neq \alpha[\pi']\langle \overrightarrow{v} \rangle, \gamma \quad \ell \neq \mathscr{R}[\alpha^\star[\pi'](\overrightarrow{v}), \gamma]}{(\alpha[\pi]\langle \overrightarrow{e} \rangle \sigma.P, \gamma) \xrightarrow{\ell}_\varepsilon \emptyset} \textbf{ Out-F2}$$

$$\frac{[\![\pi_2[\overrightarrow{v}/\overrightarrow{x}]]\!]_{\gamma_2} = \pi_2' \quad \gamma_1 \models \pi_2' \quad \gamma_2 \models \pi_1 \quad \mathbf{p} = \sigma[\overrightarrow{v}/\overrightarrow{x}](\gamma_2) \quad \varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle}{(\alpha[\pi_2](\overrightarrow{x}) \sigma.P, \gamma_2) \xrightarrow{\alpha[\pi_1](\overrightarrow{v}), \gamma_1}_\varepsilon \mu_p(\gamma_1, \gamma_2, \alpha) \cdot (P[\overrightarrow{v}/\overrightarrow{x}], \mathbf{p})} \textbf{ In}$$

$$\frac{}{(\alpha[\pi_2](\overrightarrow{x}) \sigma.P, \gamma_2) \xrightarrow{\mathscr{R}[\beta^\star[\pi_1](\overrightarrow{v}), \gamma_1]}_\varepsilon [(\alpha[\pi_2](\overrightarrow{x}) \sigma.P, \gamma_2) \mapsto 1]} \textbf{ In-F1}$$

$$\frac{[\![\pi_2[\overrightarrow{v}/\overrightarrow{x}]]\!]_{\gamma_2} = \pi_2' \quad (\gamma_1 \not\models \pi_2' \text{ or } \gamma_2 \not\models \pi_1)}{(\alpha[\pi_2](\overrightarrow{x}) \sigma.P, \gamma_2) \xrightarrow{\alpha[\pi_1](\overrightarrow{v}), \gamma_1}_\varepsilon \emptyset} \textbf{ In-F2} \quad \frac{\ell \neq \alpha[\pi_1](\overrightarrow{v}), \gamma_1 \quad \ell \neq \mathscr{R}[\beta^\star[\pi_1](\overrightarrow{v}), \gamma_1]}{(\alpha[\pi_2](\overrightarrow{x}) \sigma.P, \gamma_2) \xrightarrow{\ell}_\varepsilon \emptyset} \textbf{ In-F3}$$

$$\frac{(P, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}_1 \quad (Q, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}_2 \quad \ell \neq \mathscr{R}[\alpha^\star[\pi'](\overrightarrow{v}), \gamma]}{(P + Q, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}_1 \oplus \mathscr{C}_2} \textbf{ Plus}$$

$$\frac{(P, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi'](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}_1 \quad (Q, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi'](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}_2}{(P + Q, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi'](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}_1 + \mathscr{C}_2} \textbf{ Plus-F1}$$

$$\frac{(P, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}_1 \quad (Q, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}_2 \quad \ell \neq \mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}{(P|Q, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}_1 | Q \oplus P | \mathscr{C}_2} \textbf{ Par}$$

$$\frac{(P, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}_1 \quad (Q, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}_2}{(P|Q, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}_1 | \mathscr{C}_2} \textbf{ Par-F1} \quad \frac{A \triangleq P \quad (P, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}}{(A, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}} \textbf{ Rec}$$

$$\frac{\gamma \models \pi \quad (P, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C} \quad \ell \neq \mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}{([\pi]P, \gamma) \xrightarrow{\ell}_\varepsilon \mathscr{C}} \textbf{ Guard} \quad \frac{\gamma \models \pi \quad (P, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}_\varepsilon \mathscr{C}}{([\pi]P, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}_\varepsilon [\pi]\mathscr{C}} \textbf{ Guard-F1}$$

$$\frac{\gamma \not\models \pi \quad \ell \neq \mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}{([\pi]P, \gamma) \xrightarrow{\ell}_\varepsilon \emptyset} \textbf{ Guard-F2} \quad \frac{\gamma \not\models \pi}{([\pi]P, \gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\overrightarrow{v}), \gamma]}_\varepsilon [([\pi]P, \gamma) \mapsto 1]} \textbf{ Guard-F3}$$

Table 2: Operational semantics of components (Part 2)

$$\frac{}{\mathbf{0} \xrightarrow{\ell}_\varepsilon \emptyset}\ \textbf{Zero} \qquad \frac{(P,\gamma) \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1 \quad (P,\gamma) \xrightarrow{\mathscr{R}[\alpha^\star[\pi](\vec{\nu}),\gamma]}_\varepsilon \mathscr{N}_2}{(P,\gamma) \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \frac{\mathscr{N}_1 \oplus \mathscr{N}_2}{\oplus\mathscr{N}_1 + \oplus\mathscr{N}_2}}\ \textbf{Comp-B-In}$$

$$\frac{(P,\gamma) \xrightarrow{\ell}_\varepsilon \mathscr{N} \quad \ell \neq \mathscr{R}[\alpha^\star[\pi](\vec{\nu}),\gamma]}{(P,\gamma) \xrightarrow{\ell}_\varepsilon \mathscr{N}}\ \textbf{Comp} \qquad \frac{N_1 \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1 \quad N_2 \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_2}{N_1 \parallel N_2 \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1 \parallel \mathscr{N}_2}\ \textbf{B-In-Sync}$$

$$\frac{N_1 \xrightarrow{\alpha^\star[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_1^o \quad N_1 \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1^i \quad N_2 \xrightarrow{\alpha^\star[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_2^o \quad N_2 \xrightarrow{\alpha^\star[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_2^i}{N_1 \parallel N_2 \xrightarrow{\alpha^\star[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon (\mathscr{N}_1^o \parallel \mathscr{N}_2^i) \oplus (\mathscr{N}_1^i \parallel \mathscr{N}_2^o)}\ \textbf{B-Sync}$$

$$\frac{N_1 \xrightarrow{\alpha[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_1 \quad N_2 \xrightarrow{\alpha[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_2}{N_1 \parallel N_2 \xrightarrow{\alpha[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_1 \parallel N_2 \oplus N_1 \parallel \mathscr{N}_2}\ \textbf{Out-Sync} \qquad \frac{N_1 \xrightarrow{\alpha[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1 \quad N_2 \xrightarrow{\alpha[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_2}{N_1 \parallel N_2 \xrightarrow{\alpha[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1 \parallel N_2 \oplus N_1 \parallel \mathscr{N}_2}\ \textbf{In-Sync}$$

$$\frac{\begin{array}{ccc} N_1 \xrightarrow{\tau[\alpha[\pi]\langle\vec{\nu}\rangle,\gamma]}_\varepsilon \mathscr{N}_1^s & N_1 \xrightarrow{\alpha[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_1^o & N_1 \xrightarrow{\alpha[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_1^i \\ N_2 \xrightarrow{\tau[\alpha[\pi]\langle\vec{\nu}\rangle,\gamma]}_\varepsilon \mathscr{N}_2^s & N_2 \xrightarrow{\alpha[\pi]\langle\vec{\nu}\rangle,\gamma}_\varepsilon \mathscr{N}_2^o & N_2 \xrightarrow{\alpha[\pi](\vec{\nu}),\gamma}_\varepsilon \mathscr{N}_2^i \end{array}}{N_1 \parallel N_2 \xrightarrow{\tau[\alpha[\pi]\langle\vec{\nu}\rangle,\gamma]}_\varepsilon \frac{(\mathscr{N}_1^s \parallel N_2)\cdot\oplus\mathscr{N}_1^i}{\oplus\mathscr{N}_1^i + \oplus\mathscr{N}_2^i} \oplus \frac{(N_1 \parallel \mathscr{N}_2^s)\cdot\oplus\mathscr{N}_2^i}{\oplus\mathscr{N}_1^i + \oplus\mathscr{N}_2^i} \oplus \frac{(\mathscr{N}_1^o \parallel \mathscr{N}_2^i)}{\oplus\mathscr{N}_1^i + \oplus\mathscr{N}_2^i} \oplus \frac{(\mathscr{N}_1^i \parallel \mathscr{N}_2^o)}{\oplus\mathscr{N}_1^i + \oplus\mathscr{N}_2^i}}\ \textbf{Sync}$$

Table 3: Operational semantics of collective

Rule **Rec** is standard. The behaviour of $([\pi]P,\gamma)$ is regulated by rules **Guard**, **Guard-F1**, **Guard-F2** and **Guard-F3**. The first two rules state that $([\pi]P,\gamma)$ behaves exactly like (P,γ) when γ satisfies predicate π. However, in the first case the *guard* is removed when a transition is performed. In contrast, the *guard* still remains active after the transition when a broadcast input is refused. This is similar to what we consider for the rule **Plus-F1** and models the fact that broadcast input refusals do not remove *dynamic operators*. In rule **Guard-F1** we let $[\pi]\mathscr{C}$ denote the function that maps each term of the form $([\pi]P,\gamma)$ to $\mathscr{C}((P,\gamma))$ and any other term to 0, for each $\mathscr{C} : \text{COMP} \to \mathbb{R}_{\geq 0}$. Rules **Guard-F2** and **Guard-F3** state that no component can be reached from $([\pi]P,\gamma)$ and all the broadcast messages are refused when γ does not satisfy predicate π.

3.2 Operational semantics of collective

The operational semantics of a *collective* is defined via the transition relation $\to_\varepsilon \subseteq \text{COL} \times \text{LAB} \times [\text{COL} \to \mathbb{R}_{\geq 0}]$. This relation is formally defined in Table 3. We use a straightforward adaptation of the notations introduced in the previous section.

Rules **Zero**, **Comp-B-In** and **Comp** describe the behaviour of the single component at the level of collective. Rule **Zero** is similar to rule **Nil** of Table 1 and states that inactive component **0** cannot perform any action. Rule **Comp-B-In** states that the result of a *broadcast input* of a component at the level of *collective* is obtained by combining (summing) the transition at the level of *components* labelled $\alpha^\star[\pi](\vec{\nu}),\gamma$ with the one labelled $\mathscr{R}[\alpha^\star[\pi](\vec{\nu}),\gamma]$. This value is then renormalised to obtain a probability distribution. There we use $\oplus\mathscr{N}$ to denote $\sum_{N\in\text{COL}} \mathscr{N}(N)$. The renormalisation guarantees a reasonable computation of *broadcast output* synchronisation rates (see comments on rule **B-Sync** below).

Note that each component can always perform a *broadcast input* at the level of collective. However, we are not able to observe if the message has been received or not. Moreover, thanks to renormalisation, if $C \xrightarrow{\alpha^\star[\pi](\overrightarrow{v}),\gamma}_\varepsilon \mathscr{N}$ then $\oplus\mathscr{N} = 1$, i.e. $\mathscr{N}$ is a probability distribution over COL. Rule **Comp** simply states that for the single component $C \neq \mathbf{0}$ all the transition labels that are not a *broadcast input*, the relation $\xrightarrow{\ell}_\varepsilon$ coincides with the relation $\xrightarrow{\ell}_\varepsilon$.

Rules **B-In-Sync** and **B-Sync** describe broadcast synchronisation. The former states that two collectives N_1 and N_2 that operate in parallel synchronise while performing a broadcast input. This models the fact that the input can be potentially received by both of the collectives. In this rule we let $\mathscr{N}_1 \parallel \mathscr{N}_2$ denote the function associating the value $\mathscr{N}_1(N_1)\cdot\mathscr{N}_2(N_2)$ with each term of the form $N_1 \parallel N_2$ and 0 with all the other terms. We can observe that if $N \xrightarrow{\alpha^\star[\pi](\overrightarrow{v}),\gamma}_\varepsilon \mathscr{N}$ then, as we have already observed for rule **Comp-B-In**, $\oplus\mathscr{N} = 1$ and $\mathscr{N}$ is in fact a probability distribution over COL.

Rule **B-Sync** models the synchronisation consequent of a *broadcast output* performed at the level of a collective. For each $\mathscr{N}_1 : \text{COL} \to \mathbb{R}_{\geq 0}$ and $\mathscr{N}_2 : \text{COL} \to \mathbb{R}_{\geq 0}$, $\mathscr{N}_1 \oplus \mathscr{N}_2$ denotes the function that maps each term N to $\mathscr{N}_1(N) + \mathscr{N}_2(N)$.

At the level of collective a transition labelled $\alpha^\star[\pi]\langle\overrightarrow{v}\rangle,\gamma$ identifies the execution of a broadcast output. When a collective of the form $N_1 \parallel N_2$ is considered, the result of these kinds of transitions must be computed (in the FUTS style) by considering:

- the broadcast output emitted from N_1, obtained by the transition $N_1 \xrightarrow{\alpha^\star[\pi]\langle\overrightarrow{v}\rangle,\gamma}_\varepsilon \mathscr{N}_1^o$
- the broadcast input received by N_1, obtained by the transition $N_1 \xrightarrow{\alpha^\star[\pi](\overrightarrow{v}),\gamma}_\varepsilon \mathscr{N}_1^i$
- the broadcast output emitted from N_2, obtained by the transition $N_2 \xrightarrow{\alpha^\star[\pi]\langle\overrightarrow{v}\rangle,\gamma}_\varepsilon \mathscr{N}_2^o$
- the broadcast input received by N_2, obtained by the transition $N_2 \xrightarrow{\alpha^\star[\pi](\overrightarrow{v}),\gamma}_\varepsilon \mathscr{N}_2^i$

Note that the first synchronises with the last to obtain $\mathscr{N}_1^o \parallel \mathscr{N}_2^i$, while the second synchronises with the third to obtain $\mathscr{N}_1^i \parallel \mathscr{N}_2^o$. The result of such synchronisations are summed to model the *race condition* between the broadcast outputs performed within N_1 and N_2 respectively. We have to remark that above $\mathscr{N}_1^o$ (resp. $\mathscr{N}_2^o$) is $\emptyset$ when N_1 (resp. N_2) is not able to perform any broadcast output. Moreover, the label of a broadcast synchronisation is again a *broadcast output*. This allows further synchronisations in a derivation. Finally, it is easy to see that the total rate of a broadcast synchronisation is equal to the total rate of *broadcast outputs*. This means that the number of receivers does not affect the rate of a broadcast that is only determined by the number of senders.

Rules **Out-Sync**, **In-Sync** and **Sync** control the unicast synchronisation. Rule **Out-Sync** states that a collective of the form $N_1 \parallel N_2$ performs a *unicast output* if this is performed either in N_1 or in N_2. This is rendered in the operational semantics as an interleaving rule, where for each $\mathscr{N} : \text{COL} \to \mathbb{R}_{\geq 0}$, $\mathscr{N} \parallel N_2$ denotes the function associating $\mathscr{N}(N_1)$ with each collective of the form $N_1 \parallel N_2$ and 0 with all other collectives. Rule **In-Sync** is similar to **Out-Sync**. However, it considers *unicast input*.

Finally, rule **Sync** regulates the *unicast synchronisations* and generates transitions with labels of the form $\tau[\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma]$. This is the result of a synchronisation between transitions labelled $\alpha[\pi](\overrightarrow{v}),\gamma$, i.e. an input, and $\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma$, i.e. an output.

In rule **Sync**, $\mathscr{N}_k^s$, $\mathscr{N}_k^o$ and $\mathscr{N}_k^i$ denote the result of synchronisation ($\tau[\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma]$), unicast output ($\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma$) and unicast input ($\alpha[\pi](\overrightarrow{v}),\gamma$) within N_k ($k = 1,2$), respectively. The result of a transition labelled $\tau[\alpha[\pi]\langle\overrightarrow{v}\rangle,\gamma]$ is therefore obtained by combining:

- the synchronisations in N_1 with N_2: $\mathscr{N}_1^s \parallel N_2$;

$$\frac{\rho(\gamma_g,N)=\varepsilon=\langle\mu_r,\mu_p,\mu_u\rangle \quad N \xrightarrow{\alpha^\star[\pi]\langle\overrightarrow{\nu}\rangle,\gamma}_{\varepsilon} \mathscr{N} \quad \mu_u(\gamma_g,\alpha^\star)=(\sigma,N')}{N \textbf{ in } (\gamma_g,\rho) \xmapsto{\alpha^\star[\pi]\langle\overrightarrow{\nu}\rangle,\gamma} \mathscr{N} \parallel N' \textbf{ in } (\sigma(\gamma_g),\rho)} \textbf{ Sys-B}$$

$$\frac{\rho(\gamma_g,N)=\varepsilon=\langle\mu_r,\mu_p,\mu_u\rangle \quad N \xrightarrow{\tau[\alpha[\pi]\langle\overrightarrow{\nu}\rangle,\gamma]}_{\varepsilon} \mathscr{N} \quad \mu_u(\gamma_g,\alpha)=(\sigma,N')}{N \textbf{ in } (\gamma_g,\rho) \xmapsto{\tau[\alpha[\pi]\langle\overrightarrow{\nu}\rangle,\gamma]} \mathscr{N} \parallel N' \textbf{ in } (\sigma(\gamma_g),\rho)} \textbf{ Sys}$$

Table 4: Operational Semantics of Systems.

- the synchronisations in N_2 with N_1: $N_1 \parallel \mathscr{N}_2^s$;
- the output performed by N_1 with the input performed by N_2: $\mathscr{N}_1^o \parallel \mathscr{N}_2^i$;
- the input performed by N_1 with the output performed by N_2: $\mathscr{N}_1^i \parallel \mathscr{N}_2^o$.

To guarantee a correct computation of synchronisation rates, the first two addendi are renormalised by considering inputs performed in N_2 and N_1 respectively. This, on one hand, guarantees that the total rate of synchronisation $\tau[\alpha[\pi]\langle\overrightarrow{\nu}\rangle,\gamma]$ does not exceed the *output capacity*, i.e. the total rate of $\alpha[\pi]\langle\overrightarrow{\nu}\rangle,\gamma$ in N_1 and N_2. On the other hand, since synchronisation rates are renormalised during the derivation, it also ensures that parallel composition is associative [7].

3.3 Operational semantics of systems

The operational semantics of systems is defined via the transition relation $\mapsto \subseteq \text{SYS} \times \text{LAB} \times [\text{SYS} \to \mathbb{R}_{\geq 0}]$ that is formally defined in Table 4. Only synchronisations are considered at the level of systems.

The first rule is **Sys-B**. This rule states that a system of the form N **in** (γ_g,ρ) can perform a *broadcast output* when the collective N, under the environment evaluation $\varepsilon=\langle\mu_r,\mu_p,\mu_u\rangle=\rho(\gamma_g,N)$, can evolve at the level of collective with the label $\alpha^\star[\pi]\langle\overrightarrow{\nu}\rangle,\gamma$ to $\mathscr{N}$. After the transition, the global store is updated and a new collective can be created according to function μ_u. In rule **Sys-B** the following notations are used. For each collective N_2, $\mathscr{N}:\text{COL}\to\mathbb{R}_{\geq 0}$, $\mathscr{S}:\text{SYS}\to\mathbb{R}_{\geq 0}$ and $\mathbf{p}\in \mathit{Dist}(\Gamma)$ we let $\mathscr{N}$ **in** $(\mathbf{p},\rho)$ denote the function mapping each system N **in** (γ,ρ) to $\mathscr{N}(N)\cdot\mathbf{p}(\gamma)$. The second rule is **Sys** that is similar to **Sys-B** and regulates unicast synchronisations.

4 CARMA at work

In this section we will use CARMA to model a *bike sharing* system [6, 17]. These systems are a recent, and increasingly popular, form of public transport in urban areas. As a resource-sharing system with large numbers of independent users altering their behaviour due to pricing and other incentives, they are a simple instance of a collective adaptive system, and hence a suitable case study to exemplify the CARMA language.

The idea in a bike sharing system is that bikes are made available in a number of stations that are placed in various areas of a city. Users that plan to use a bike for a short trip can pick up a bike at a suitable origin station and return it to any other station close to their planned destination. One of the major issues in bike sharing systems is the availability and distribution of resources, both in terms of available bikes at the stations and in terms of available empty parking places in the stations, where users will park the bikes after using them.

In our scenario we assume that the city is partitioned in homogeneous zones and that all the *stations* in the same zone can be equivalently used by any user in that zone. Below, we let $\{z_0,\ldots,z_n\}$ be the n zones in the city, each of which contains k parking stations.

Each parking station is modelled in CARMA via a component of the form:

$$(\ G|R\ ,\{\mathsf{zone}=\ell,\mathsf{bikes}=i,\mathsf{slots}=j\})$$

where

- zone is the attribute identifying the zone where the parking station is located;
- bikes is the attribute used to count the number of available bikes;
- slots is the attribute containing the total number of parking slots in the parking station.

Processes G and R, which model the procedure to *get* and *return* a bike in the parking station, respectively, are defined as follow:

$$G \triangleq [\mathsf{bikes}>0]\ \mathsf{get}[\mathsf{zone}=\mathsf{this.zone}]\langle\bullet\rangle\{\mathsf{bikes}\leftarrow\mathsf{bikes}-1\}.G$$

$$R \triangleq [\mathsf{slots}>\mathsf{bikes}]\ \mathsf{ret}[\mathsf{zone}=\mathsf{this.zone}]\langle\bullet\rangle\{\mathsf{bikes}\leftarrow\mathsf{bikes}+1\}.R$$

Process G, when the value of attribute bikes is greater than 0, executes the *unicast output* with action type get that potentially involves components satisfying the predicate zone = this.zone, i.e. the ones that are located in the same zone[3]. When the output is executed the value of the attribute bikes is decreased by one to model the fact that one bike has been retrieved from the parking station.

Process R is similar. It executes the *unicast output* with action type ret that potentially involves components satisfying predicate zone = this.zone. This action can be executed only when there is at least one parking slot available, i.e. when the value of attribute bikes is less than the value of attribute slots. When the output considered above is executed, the value of attribute bikes is increased by one to model the fact that one bike has been returned in the parking station.

Users, who can be either *bikers* or *pedestrians*, are modelled via components of the form:

$$(Q,\{\mathsf{zone}=\ell\})$$

where zone is the attribute indicating where the user is located, while Q models the current state of the user and can be one of the following processes:

$$\begin{aligned} B &\triangleq \mathsf{move}^\star[\bot]\langle\bullet\rangle\{\mathsf{zone}\leftarrow U(z_0,\ldots,z_n)\}.B \\ &\quad + \ \mathsf{stop}^\star[\bot]\langle\bullet\rangle.WS \\ WS &\triangleq \mathsf{ret}[\mathsf{zone}=\mathsf{this.zone}](\bullet).P \\ P &\triangleq \mathsf{go}^\star[\bot]\langle\bullet\rangle.WS \\ WB &\triangleq \mathsf{get}[\mathsf{zone}=\mathsf{this.zone}](\bullet).B \end{aligned}$$

Process B represents a *biker*. When a user is in this state (s)he can either *move* from the current zone to another zone or *stop* to return the bike to a parking station. These activities are modelled with the

[3] Here we use $\bullet$ to denote the unit value.

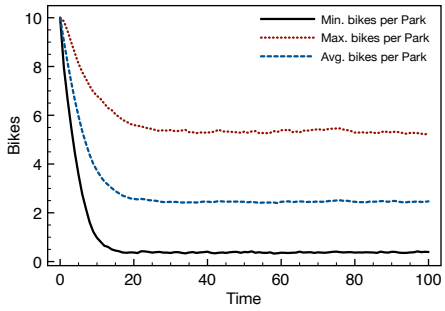

Figure 1: Simulation of bike scenario.

execution of a broadcast output via action types move and stop, respectively. Note that in both of these cases, the predicate used to identify the target of the actions is $\bot$, denoting the value *false*. This means that neither of the two actions actually synchronise with any component (since no component satisfies $\bot$). This kind of interaction is used in CARMA to model *spontaneous actions*, i.e. actions that render the execution of an activity and that do no require synchronisation. After the broadcast $\mathsf{move}^\star$ the value of attribute zone is updated by randomly selecting the next zone in $\{z_0,\ldots,z_n\}$. With $\{\mathsf{zone} \leftarrow U(z_0,\ldots,z_n)\}$ we denote the update σ such that $\sigma(\gamma)$ is the probability distribution giving probability $\frac{1}{n}$ to each store $\gamma[\mathsf{zone} \leftarrow z_i]$. This update models a random movement of the user among the city zones.

When process B executes broadcast $\mathsf{stop}^\star$, it evolves to process WS. This process models a user who is waiting for a parking slot. This process executes an input over ret. This models the fact that the user has found a parking station with an available parking slot in their zone. After the execution of this input process P is executed. The latter component definition models a *pedestrian user*. The user remains in this state until the *spontaneous action* $\mathsf{go}^\star$ is performed. After that it evolves to process WB which models a user waiting for a bike. The behaviour of WB is similar to that of WS described above.

Using a custom-built prototype simulator, we are able to simulate this modelled scenario. The output on one simulation run is presented in Figure 1. In the graph we show the minimum, average and maximum number of bikes in one zone of the city. We consider a scenario with four zones each containing four parking stations. The total number of users is 150.

5 Conclusions

We have presented CARMA, a new stochastic process algebra for the representation of systems developed in the CAS paradigm. The language offers a rich set of communication primitives, and the use of attributes, captured in a store associated with each component, allows attribute-based communication. For most CAS systems we anticipate that one of the attributes will be the location of the agent and thus it is straightforward to capture systems in which, for example, there is a limited scope of communication, or restriction to only interact with components that are co-located. As demonstrated in the case study presented in Section 4, attributes can also be used to capture the "state" of a component, such as the available number of bikes/slots at a bike station.

CARMA reflects the experience that we have gained through earlier languages such as SCEL [8] and PALOMA [9]. Compared with SCEL, the representation of knowledge here is more abstract, and not designed for detailed reasoning during the evolution of the model. This reflects the different objectives of

the languages. Whilst SCEL is designed to support the programming of autonomic computing systems, the primary focus of CARMA is quantitative analysis. In stochastic process algebras such as PEPA, MTIPP and EMPA, data is typically abstracted away, and the influence of data on behaviour is captured only stochastically. When the data is important to differentiate behaviour it must be implicitly encoded in the state of components. In the context of CAS we wish to support attribute-based communication to reflect the flexible and dynamic interactions that occur in such systems. Thus it is not possible to entirely abstract from data. On the other hand, the level of abstraction means that choices within the system will be captured stochastically rather than through the rich policies for reasoning offered by SCEL. We believe that this offers a reasonable compromise between expressiveness and tractability. Another key feature of CARMA is the inclusion of an explicit environment in which components interact. In PALOMA there was a rudimentary form of environment, termed the *perception function* but this proved cumbersome to use, and it could not itself be influenced by the behaviour of the components. In CARMA, in contrast, the environment not only modulates the rates and probabilities related to interactions between components, it can also itself evolve at runtime, due to feedback from the collective.

The focus of this paper has been the discrete semantics in the structured operational style of FUTS [7], but in future work we plan to develop differential semantics in the style of [16]. This latter approach will be essential in order to support quantitative analysis of CAS systems of realistic scale, but it may not be possible to encompass the full rich set of language features of CARMA with such efficient analysis. Further work is needed to investigate this issue, and which language features can be supported for the various forms of quantitative analysis available. Additional work involves the development of an appropriate high-level language for designers of CAS which will be mapped to the process algebra, and hence will enable qualitative and quantitive analysis of CAS during system development by enabling a design workflow and analysis pathway. The intention of this high-level language is not to add to the expressiveness of CARMA, which we believe to be well-suited to capturing the behaviour of CAS, but rather to ease the task of modelling for users who are unfamiliar with process algebra and similar formal notations.

Acknowledgements

This work is partially supported by the EU project QUANTICOL, 600708. This research has also been partially funded by the German Research Council (DFG) as part of the Cluster of Excellence on Multimodal Computing and Interaction at Saarland University.

References

[1] Yehia Abd Alrahman, Rocco De Nicola, Michele Loreti, Francesco Tiezzi & Roberto Vigo (2015): *A Calculus for Attribute-based Communication*. In: *Proceedings of SAC 2015*. To appear.

[2] Marco Bernardo & Roberto Gorrieri (1998): *A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time*. *Theoretical Computer Science* 202(1-2), pp. 1–54.

[3] H.C. Bohnenkamp, P.R. D'Argenio, H. Hermanns & J-P. Katoen (2006): *MODEST: A Compositional Modeling Formalism for Hard and Softly Timed Systems*. *IEEE Trans. Software Eng.* 32(10), pp. 812–830, doi:10.1109/TSE.2006.104. Available at `http://doi.ieeecomputersociety.org/10.1109/TSE.2006.104`.

[4] Luca Bortolussi & Alberto Policriti (2010): *Hybrid dynamics of stochastic programs*. *Theor. Comput. Sci.* 411(20), pp. 2052–2077, doi:10.1016/j.tcs.2010.02.008.

[5] Federica Ciocchetta & Jane Hillston (2009): *Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems*. Theoretical Computer Science 410(33), pp. 3065–3084.

[6] Paola De Maio (2009): *Bike-sharing: Its History, Impacts, Models of Provision, and Future.* Journal of Public Transportation 12(4), pp. 41–56.

[7] Rocco De Nicola, Diego Latella, Michele Loreti & Mieke Massink (2013): *A uniform definition of stochastic process calculi*. ACM Comput. Surv. 46(1), p. 5, doi:10.1145/2522968.2522973.

[8] Rocco De Nicola, Michele Loreti, Rosario Pugliese & Francesco Tiezzi (2014): *A Formal Approach to Autonomic Systems Programming: The SCEL Language*. TAAS 9(2), p. 7, doi:10.1145/2619998.

[9] Cheng Feng & Jane Hillston (2014): *PALOMA: A Process Algebra for Located Markovian Agents*. In: Quantitative Evaluation of Systems - 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings, Lecture Notes in Computer Science 8657, Springer, pp. 265–280, doi:10.1007/978-3-319-10696-0_22.

[10] Daniel T Gillespie (1976): *A general method for numerically simulating the stochastic time evolution of coupled chemical reactions*. Journal of Computational Physics 22(4), pp. 403 – 434, doi:http://dx.doi.org/10.1016/0021-9991(76)90041-3.

[11] Holger Hermanns, Ulrich Herzog & Joost-Pieter Katoen (2002): *Process algebra for performance evaluation*. Theor. Comput. Sci. 274(1-2), pp. 43–87, doi:10.1016/S0304-3975(00)00305-4. Available at `http://dx.doi.org/10.1016/S0304-3975(00)00305-4`.

[12] Holger Hermanns & Michael Rettelbach (1994): *Syntax, Semantics, Equivalences and Axioms for MTIPP*. In U. Herzog & M. Rettelbach, editors: Proc. of 2nd Process Algebra and Performance Modelling Workshop.

[13] Jane Hillston (1995): *A Compositional Approach to Performance Modelling*. CUP.

[14] Diego Latella, Michele Loreti, Mieke Massink & Valerio Senni (2014): *Stochastically timed predicate-based communication primitives for autonomic computing*. In Nathalie Bertrand & Luca Bortolussi, editors: Proceedings Twelfth International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2014, Grenoble, France, 12-13 April 2014., EPTCS 154, pp. 1–16, doi:10.4204/EPTCS.154.1.

[15] Corrado Priami (1995): *Stochastic π-calculus*. The Computer Journal 38(7), pp. 578–589.

[16] Mirco Tribastone, Stephen Gilmore & Jane Hillston (2012): *Scalable Differential Analysis of Process Algebra Models*. IEEE Transactions on Software Engineering 38(1), pp. 205–219.

[17] Wikipedia (2013): *Bicycle sharing system — Wikipedia, The Free Encyclopedia*. Available at `http://en.wikipedia.org/w/index.php?title=Bicycle_sharing_system&oldid=573165089`. [Online; accessed 17-September-2013].